

ASSEGNAIMENTO DI RECUPERO: BBVS

Informatica, Corso di Laurea in Fisica, Università di Pisa

AA 2019/20

1 Bubble Breaker Versus

Il gioco Bubble Breaker si basa su una griglia, rappresentata da una matrice di caratteri di dimensione $L \times L$, dove ogni cella contiene un carattere (o colore), oppure è *vuota*.

Vi sono h colori in totale (è un input del gioco), rappresentati dai numeri $0, 1, \dots, h-1$, che corrispondono alle celle *piene*.

Ad esempio, se $h = 3$, i possibili colori corrispondono ai numeri $0, 1$, e 2 . Il valore delle celle vuote è definito dalla macro `EMPTY` in `bbvs.h`.

Le regole del gioco sono le seguenti:

- La board del gioco è una matrice $L \times L$ di numeri *interi*, dove la cella `board[0][0]` corrisponde alla cella in alto a sinistra.
- I parametri della partita sono n, h, t : La board avrà dimensione $n \times n$, e le celle h possibili valori compresi tra 0 e $h-1$. t è il punteggio da raggiungere per vincere.
- All'inizio di una nuova partita, la board viene inizializzata con valori casuali.
- La partita consiste nell'alternarsi di un turno del giocatore umano, e un turno del computer, che effettuano mosse sulla stessa board finché uno dei due non raggiunge il punteggio t (il punteggio di un giocatore è la somma dei punteggi di tutte le mosse effettuate da esso). Dopo ogni mossa, la board viene riempita.
- Nel suo turno, il giocatore può effettuare una mossa selezionando una cella i, j o decidere di interrompere la partita. Nel secondo caso, la configurazione della board è salvata nel file "`partita-nome.txt`" (vedi descrizione delle funzioni).
- Le mosse sono solo di tipo "bubble" (vedi descrizione delle funzioni), e sono valide solo se la bubble è composta da almeno due celle.
- La partita termina quando il giocatore o il computer raggiungono il punteggio t , o se non ci sono mosse possibili (in ogni caso, vince chi ha il punteggio più alto al termine della partita).

2 Regole di pop bubble

L'unica regola di pop è `pop_bubble`, da realizzare in `int pop_bubble (partita_t * p, int i, int j)`.

Data una cella i, j *non vuota*, contenente il valore v , la *bubble* di i, j corrisponde alla cella `board[i][j]` e tutta la porzione raggiungibile dalla cella i, j della matrice contenente lo stesso carattere.

Ogni cella è considerata vicina alla cella alla sua sinistra, destra, sopra, e sotto, ma non diagonalmente. La figura sotto fa vedere un esempio.

Una mossa è considerata valida se la bubble è composta da *almeno* due celle. Non è ammesso colpire bubble composte da una sola cella, o celle vuote.

Se la mossa è valida, `pop_bubble` esplode tutte le celle nella bubble di i, j , e applica le regole di gravità avanzate, che fanno cadere le celle piene verso il basso e compattano le colonne vuote facendole scorrere quelle non vuote verso destra (vedi descrizione delle funzioni).

La funzione restituisce il punteggio assegnato alla mossa c^2 , dove c è il numero di celle esplose dalla mossa; in caso di mossa non valida (cella vuota, bubble di dimensione 1, o coordinate non valide), la funzione restituisce il valore `ERRORE` (definito in `bbvs.h`), e non modifica la board. Vediamo un esempio:

```

0 0 0 0 0 2
1 1 0 2 0 0
1 1 0 1 0 1
2 2 0 2 0 2
1 0 0 2 0 3
3 2 0 2 2 0

```

La board 6x6
di una partita p

```

0 0 0 0 0 2
* * 0 2 0 0
*(*)0 1 0 1
2 2 0 2 0 2
1 0 0 2 0 3
3 2 0 2 2 0

```

*: celle esplose da
 $\text{pop_bubble}(p, 2, 1)$

```

0 0 0 2
0 2 0 0
0 0 0 1 0 1
2 2 0 2 0 2
1 0 0 2 0 3
3 2 0 2 2 0

```

La board dopo
 $\text{pop_bubble}(p, 2, 1)$

```

0 1 0 0 0 2
2 0 0 2 0 0
0 0 0 1 0 1
2 2 0 2 0 2
1 0 0 2 0 3
3 2 0 2 2 0

```

La board viene
riempita.

```

0 1 * * * 2
2 *(*)2 * *
* * * 1 * 1
2 2 * 2 * 2
1 * * 2 * 3
3 2 * 2 2 0

```

*: celle esplose da
 $\text{pop_bubble}(p, 1, 2)$

```

0 2 2
2 1 1
2 1 2 2
1 2 2 3
3 2 2 2 0

```

La board dopo
 $\text{pop_bubble}(p, 1, 2)$

```

2 0 1 3 0 1
3 2 0 2 1 2
2 0 1 1 2 1
0 2 3 2 2 2
1 1 2 2 0 3
0 3 2 2 2 0

```

La board viene
riempita.

```

2 0 1 3 0 1
3 2 0 2 1 2
2 0 1 1 * 1
0 2 3 * * *
1 1 * * 0 3
0 3(*)* * 0

```

*: celle esplose da
 $\text{pop_bubble}(p, 5, 2)$

Notare in particolare come le colonne vuote sono compattate spostando le piene verso destra, e come alla fine non ci sono mosse valide disponibili in quanto tutte le bubble rimaste sono da 1 cella. Tuttavia, in questa modalità la board viene riempita dopo ogni mossa.

3 Salvataggio/caricamento della partita

Se il giocatore *nome* decide di interrompere la partita, viene salvata la partita nel file “partita-*nome*.txt” (es. “partita-pippo.txt”, chiamando *salva(pippo, p)*).

Il file deve contenere tutte le informazioni necessarie a ripristinare la partita: in particolare il file dovrà contenere una riga contenente i valori n, h, t , e i punteggi effettuati dal giocatore e computer (separati da spazi), e a seguire la board, anche essa riga per riga con elementi separati da spazi.

Un possibile esempio di partita è nel file “partita-pippo.txt”, ottenuto tramite (dove $n = 6, h = 4, t = 100$ il giocatore *pippo* ha effettuato 9 punti e il computer 12):

```

6 4 100 9 12
0 0 0 0 0 2
1 1 0 2 0 0
1 1 0 1 0 1
2 2 0 2 0 2
1 0 0 2 0 3
3 2 0 2 2 0

```

3.1 Caricamento a inizio partita

Deve anche essere possibile caricare una partita in memoria tramite la funzione *carica(nome)*: ad esempio chiamando *carica(pippo)* la funzione deve restituire la partita salvata nel file “partita-pippo.txt”.

Questo viene utilizzato a **inizio partita**: il giocatore immette il proprio nome a inizio partita. Se esiste il file “partita-*nome*.txt” col *nome* inserito dal giocatore, la funzione *gioca* deve caricare la partita contenuta nel file e continuare a giocare sulla stessa (partendo dal turno del giocatore).

4 Cosa deve essere realizzato

Lo studente deve realizzare un file *bbvs.c* contenente le funzioni che servono ad eseguire i componenti del gioco BUBBLE BREAKER, inclusa una funzione *gioca* che permette al giocatore di giocare una partita.

I prototipi delle funzioni da realizzare (insieme alle definizioni essenziali) in *bbvs.c* si trovano nel file *bbvs.h*.

- `partita_t * crea_partita(int n, int h, int t)`: crea una nuova partita, allocando dinamicamente la memoria e inizializzandone tutti i campi opportuni, inclusa la board, e inizializza ogni sua cella con un intero nell'intervallo $0, 1, \dots, h - 1$ scelto in modo casuale. Restituisce la partita creata, o il valore `NULL` in caso di errore.
- `void distruggi_partita(partita_t * p)`: distrugge la partita data, liberando la memoria in modo appropriato.
- `void gravita (partita_t * p)`: applica le regole di gravità sulla board della partita data. Fa cadere gli elementi in celle che si trovano sopra a celle vuote verso il basso, fino ad appoggiarsi sulle celle non vuote sottostanti, o sul fondo. Successivamente compatta le colonne facendo scorrere verso destra le colonne non vuote finché possibile.
- `int pop_bubble (partita_t * p, int i, int j)`: esegue la mossa sulla cella i, j della partita p . In particolare esplode la cella i, j e tutta la sua bubble, come in figura sopra, se questa è composta da almeno due celle. Poi applica le regole di gravità. La funzione restituisce il punteggio della mossa (ovvero il quadrato del numero di celle esplose), o ERRORE in caso di mossa invalida (ad es, colpisco una cella vuota, o fuori dalla board, o una bubble da una sola cella). In caso di errore non viene fatta alcuna modifica alla partita. Nota: questa funzione NON si occupa di riempire la board.
- `int riempi (partita_t * p)`: Riempie tutte e sole le celle vuote nella board della partita p , usando la soglia h contenuta in essa (e quindi valori casuali nell'intervallo $0, 1 \dots h - 1$). Restituisce 1 se la board è stata modificata, 0 altrimenti.
- `int valida (partita_t * p)`: restituisce 1 se c'è almeno una mossa valida che può essere eseguita nella board della partita p , 0 altrimenti.
- `int salva (char * nome, partita_t * p)`: Salva lo stato della partita (come descritto sopra) nel file "partita-nome.txt", sovrascrivendolo se già esistente. Restituisce 1 se l'operazione va a buon fine e 0 altrimenti.
- `partita_t * carica (char * nome)`: Crea una partita corrispondente a quella salvata nel file "partita-nome.txt" e la restituisce. In caso di errore restituisce il valore `NULL`.
- `int gioca (int n, int h, int t)`: Gestisce un'intera partita di *bubble breaker vs* contro il computer, tramite le regole spiegate sopra, interfacciandosi con l'utente tramite il terminale. La funzione chiede immediatamente il nome del giocatore: se esistente il file relativo ripristina la partita nel file, altrimenti crea una nuova partita con i parametri n, h, t in input. Restituisce 0 se il programma termina correttamente, 1 in caso di terminazione precoce (ad es, impossibile creare la partita).

I file `*test.c` contengono dei main che usano queste funzioni ed effettuano dei test sul loro funzionamento. Tali test possono essere attivati automaticamente utilizzando il `Makefile` come specificato nel file `README`. Solo il codice che supera con successo questi test può essere consegnato.

Tuttavia, è bene ricordare che il superamento dei test non garantisce la correttezza completa della soluzione, quindi invitiamo gli studenti ad analizzare attentamente i risultati ottenuti e le stampe effettuate prima della consegna.

5 Consegna

Lo studente deve consegnare un archivio contenente i file `bbvs.c` e `gruppo.txt` (le istruzioni su come generare questo archivio sono nel file `README`).

Le funzioni realizzate devono essere adeguatamente (ma non esageratamente) commentate come discusso a lezione.

Valutazione: Viene valutato positivamente un progetto che implementa tutte le funzionalità richieste in modo adeguato seguendo gli insegnamenti del corso. E' chiaramente possibile realizzare funzioni ausiliarie nel file `bbvs.c` (ad esempio, per la stampa della board). Viene valutata positivamente l'attenzione al dettaglio (e.g. la gestione delle risorse, o la qualità dell'interfaccia utente). E' anche possibile realizzare eventuali funzionalità aggiuntive (come un computer particolarmente intelligente, o stampa della board a colori) e consegnare una breve discussione su quanto realizzato e come utilizzarlo sotto forma di file PDF. Questo però non deve modificare il normale funzionamento delle funzioni richieste dalla consegna (il codice consegnato deve in ogni caso passare i test).